Introduction

Welcome to **CS188 - Data Science Fundamentals!** This course is designed to equip you with the tools and experiences necessary to start you off on a life-long exploration of datascience. We do not assume a prerequisite knowledge or experience in order to take the course.

For this first project we will introduce you to the end-to-end process of doing a datascience project. Our goals for this project are to:

1. Familiarize you with the development environment for doing datascience
2. Get you comfortable with the python coding required to do datascience
3. Provide you with an sample end-to-end project to help you visualize the steps needed to complete a project on your own
4. Ask you to recreate a similar project on a separate dataset

In this project you will work through an example project end to end. Many of the concepts you will encounter will be unclear to you. That is OK! The course is designed to teach you these concepts in further detail. For now our focus is simply on having you replicate the code successfully and seeing a project through from start to finish.

Here are the main steps:

1. Get the data
2. Visualize the data for insights
3. Preprocess the data for your machine learning algorithm
4. Select a model and train
5. Does it meet the requirements? Fine tune the model

steps

Working with Real Data

It is best to experiment with real-data as opposed to aritifical datasets.

There are many different open datasets depending on the type of problems you might be interested in!

Here are a few data repositories you could check out:

- UCI Datasets
- Kaggle Datasets
- AWS Datasets

Submission Instructions

When you have completed this assignment please save the notebook as a PDF file and submit the assignment via Gradescope

Example Datascience Exercise

Below we will run through an California Housing example collected from the 1990's.

Setup

In [3]:

```
import sys
assert sys.version_info >= (3, 5) # python>=3.5
import sklearn
assert sklearn.__version__ >= "0.20" # sklearn >= 0.20

import numpy as np #numerical package in python
import os
%matplotlib inline
import matplotlib.pyplot as plt #plotting package

# to make this notebook's output identical at every run
np.random.seed(42)

#matplotlib magic for inline figures
%matplotlib inline
import matplotlib # plotting library
import matplotlib.pyplot as plt

# Where to save the figures
ROOT_DIR = "."
IMAGES_PATH = os.path.join(ROOT_DIR, "images")
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_name, tight_layout=True, fig_extension="png", resolution=300):
    '''
        plt.savefig wrapper. refer to
        https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.savefig.html

        Args:
            fig_name (str): name of the figrue
            tight_layout (bool): adjust subplot to fit in the figure area
            fig_extension (str): file format to save the figure in
            resolution (int): figure resolution
    '''
    path = os.path.join(IMAGES_PATH, fig_name + "." + fig_extension)
    print("Saving figure", fig_name)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

In [4]:

```
import os
import tarfile
import urllib
DATASET_PATH = os.path.join("datasets", "housing")
```

Step 1. Getting the data

Intro to Data Exploration Using Pandas

In this section we will load the dataset, and visualize different features using different types of plots.

Packages we will use:

- **Pandas:** is a fast, flexible and expressive data structure widely used for tabular and multidimensional datasets.
- **Matplotlib**: is a 2d python plotting library which you can use to create quality figures (you can plot almost anything if you're willing to code it out!)
  - other plotting libraries:seaborn, ggplot2

In [5]:

```
import pandas as pd

def load_housing_data(housing_path):
    '''
        loads housing.csv dataset stored

        Args:
            housing_path (str): path to folder containing housing datased

        Returns:
            pd.DataFrame
    '''
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

In [6]:

```
pd.DataFrame
```

Out[6]: `pandas.core.frame.DataFrame`

In [7]:

```
housing = load_housing_data(DATASET_PATH) # we load the pandas dataframe
housing.head() # show the first few elements of the dataframe
               # typically this is the first thing you do
               # to see how the dataframe looks like
```

Out[7]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	mec
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

A dataset may have different types of features

- real valued
- Discrete (integers)
- categorical (strings)

The two categorical features are essentialy the same as you can always map a categorical string/character to an integer.

In the dataset example, all our features are real valued floats, except ocean proximity which is categorical.

In [6]:
```
# to see a concise summary of data types, null values, and counts
# use the info() method on the dataframe
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [7]:
```
# you can access individual columns similarly
# to accessing elements in a python dict
housing["ocean_proximity"].head() # added head() to avoid printing many columns..
```

Out[7]:
```
0    NEAR BAY
1    NEAR BAY
2    NEAR BAY
3    NEAR BAY
4    NEAR BAY
Name: ocean_proximity, dtype: object
```

In [8]:
```
# to access a particular row we can use iloc
housing.iloc[1]
```

Out[8]:
```
longitude              -122.22
latitude                 37.86
housing_median_age        21.0
total_rooms             7099.0
total_bedrooms          1106.0
population              2401.0
households              1138.0
median_income           8.3014
median_house_value    358500.0
ocean_proximity       NEAR BAY
Name: 1, dtype: object
```

In [9]:
```
# one other function that might be useful is
# value_counts(), which counts the number of occurences
# for categorical features
housing["ocean_proximity"].value_counts()
```

Out[9]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

In [10]:
```
# The describe function compiles your typical statistics for each
# column
housing.describe()
```

Out[10]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	2(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	(

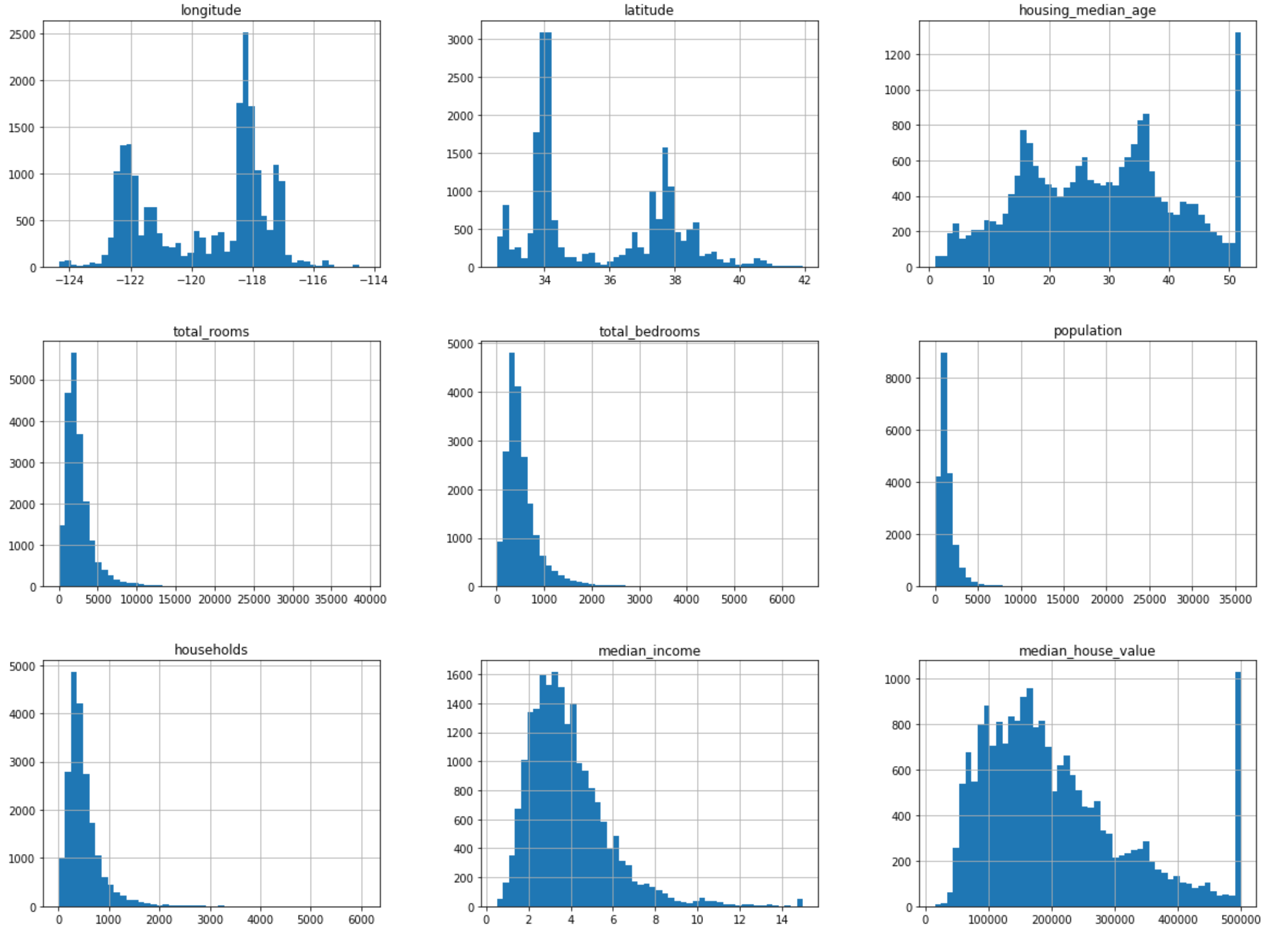
If you want to learn about different ways of accessing elements or other functions it's useful to check out the getting started section here

Step 2. Visualizing the data

Let's start visualizing the dataset

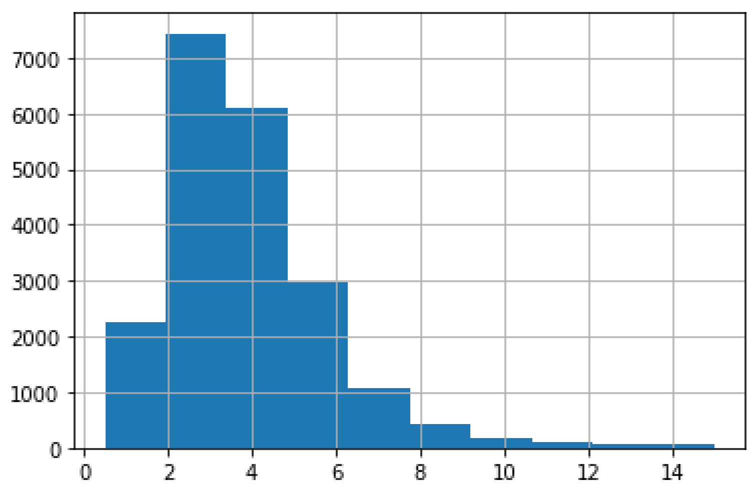
In [11]:
```
# We can draw a histogram for each of the dataframes features
# using the hist function
housing.hist(bins=50, figsize=(20,15))
# save_fig("attribute_histogram_plots")
plt.show() # pandas internally uses matplotlib, and to display all the figures
           # the show() function must be called
```

```
/mnt/c/Users/ryanr/OneDrive/Desktop/UCLA/CSM148/lib/python3.7/site-packages/pandas/plott
ing/_matplotlib/tools.py:400: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed two minor
releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```



In [18]:

```
# if you want to have a histogram on an individual feature:
housing["median_income"].hist()
plt.show()
```



We can convert a floating point feature to a categorical feature by binning or by defining a set of intervals.

For example, to bin the households based on median_income we can use the pd.cut function

In [19]:

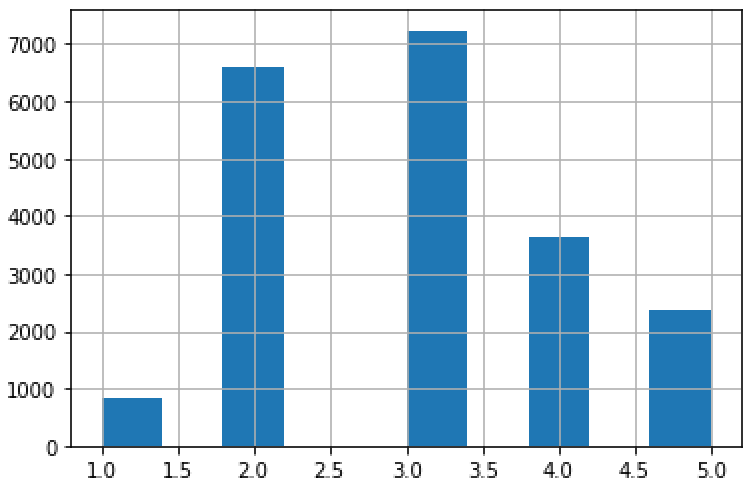
```
# assign each bin a categorical value [1, 2, 3, 4, 5] in this case.
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
```

```
                                labels=[1, 2, 3, 4, 5])

housing["income_cat"].value_counts()
```

Out[19]:
```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

In [20]:
```
housing["income_cat"].hist()
```

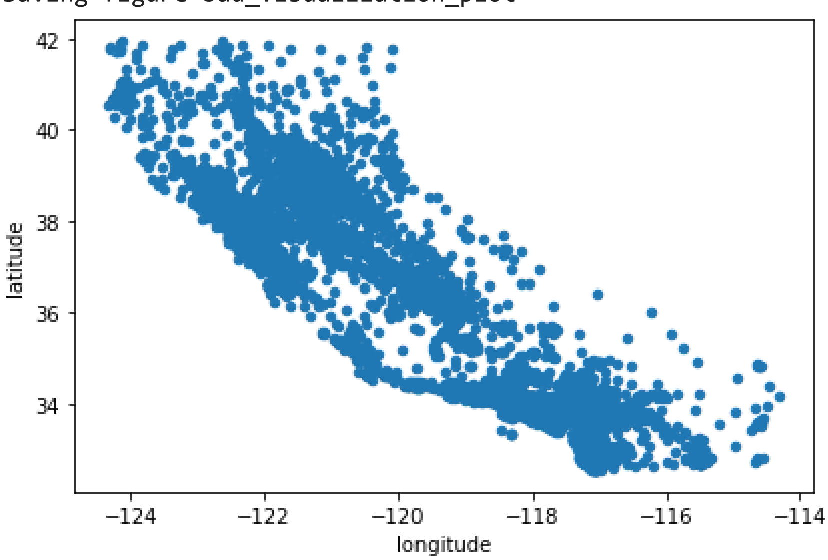
Out[20]: <AxesSubplot:>



Next let's visualize the household incomes based on latitude & longitude coordinates

In [21]:
```
## here's a not so interestting way of plotting it
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

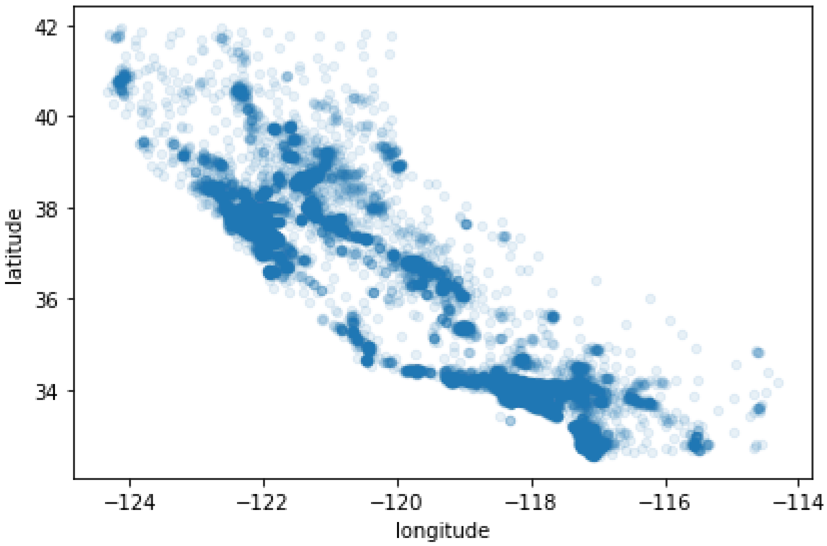
Saving figure bad_visualization_plot



In [22]:

```
# we can make it look a bit nicer by using the alpha parameter,
# it simply plots less dense areas lighter.
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
save_fig("better_visualization_plot")
```

Saving figure better_visualization_plot

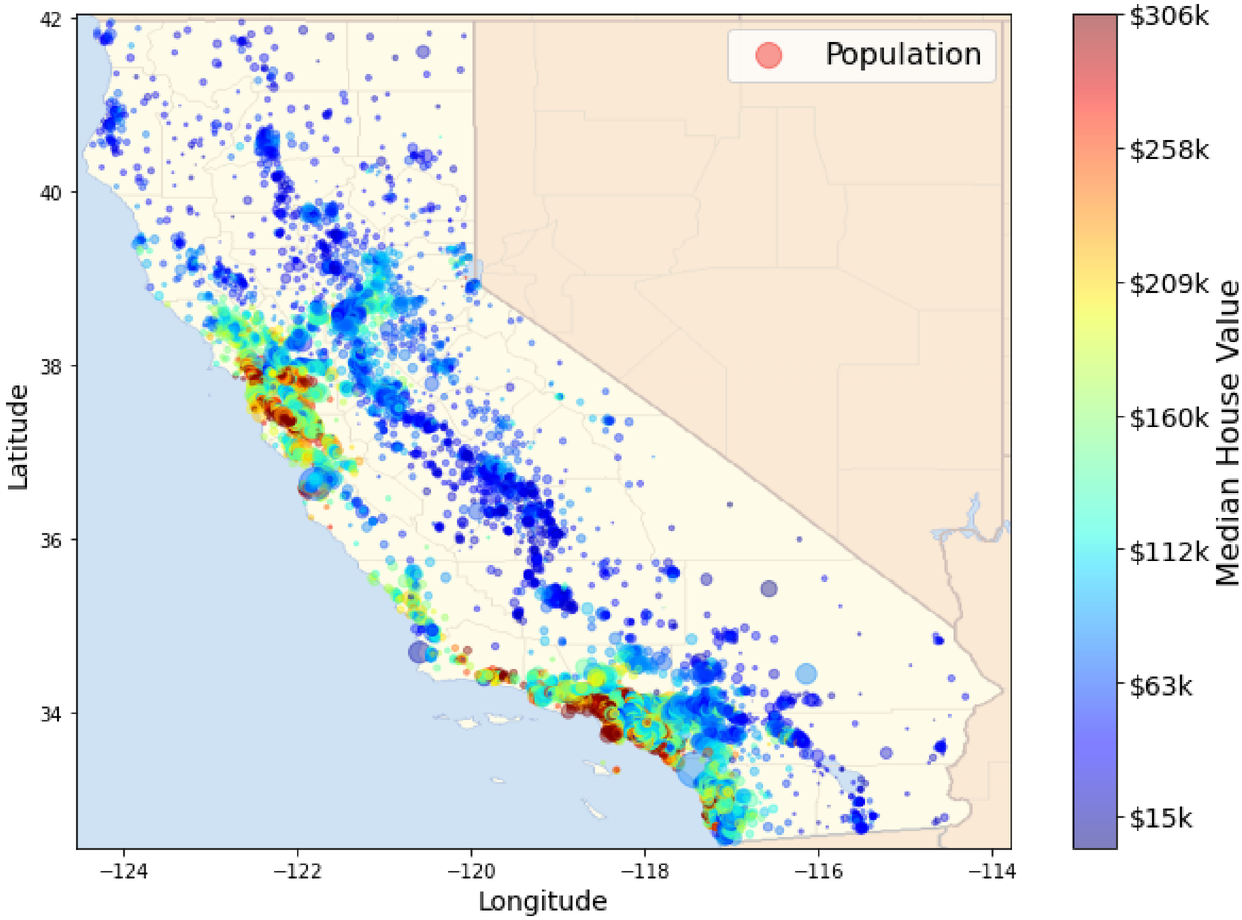


In [23]:

```
# A more interesting plot is to color code (heatmap) the dots
# based on income. The code below achieves this

# Load an image of california
images_path = os.path.join('./', "images")
os.makedirs(images_path, exist_ok=True)
filename = "california.png"

import matplotlib.image as mpimg
california_img=mpimg.imread(os.path.join(images_path, filename))
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                       s=housing['population']/100, label="Population",
                       c="median_house_value", cmap=plt.get_cmap("jet"),
                       colorbar=False, alpha=0.4,
                      )
# overlay the califronia map on the plotted scatter plot
# note: plt.imshow still refers to the most recent figure
# that hasn't been plotted yet.
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

# setting up heatmap colors based on median_house_value feature
prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cb = plt.colorbar()
cb.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14)
cb.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
save_fig("california_housing_prices_plot")
plt.show()
```

/mnt/c/Users/ryanr/OneDrive/Desktop/UCLA/CSM148/lib/python3.7/site-packages/ipykernel_la

```
uncher.py:28: UserWarning: FixedFormatter should only be used together with FixedLocator
Saving figure california_housing_prices_plot
```



Not suprisingly, we can see that the most expensive houses are concentrated around the San Francisco/Los Angeles areas.

Up until now we have only visualized feature histograms and basic statistics.

When developing machine learning models the predictiveness of a feature for a particular target of intrest is what's important.

It may be that only a few features are useful for the target at hand, or features may need to be augmented by applying certain transformations.

None the less we can explore this using correlation matrices. If you need to brush up on correlation take a look here.

In [24]:

```
corr_matrix = housing.corr() # compute the correlation matrix
```

In [25]:

```
# for example if the target is "median_house_value", most correlated features can be so
# which happens to be "median_income". This also intuitively makes sense.
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[25]:

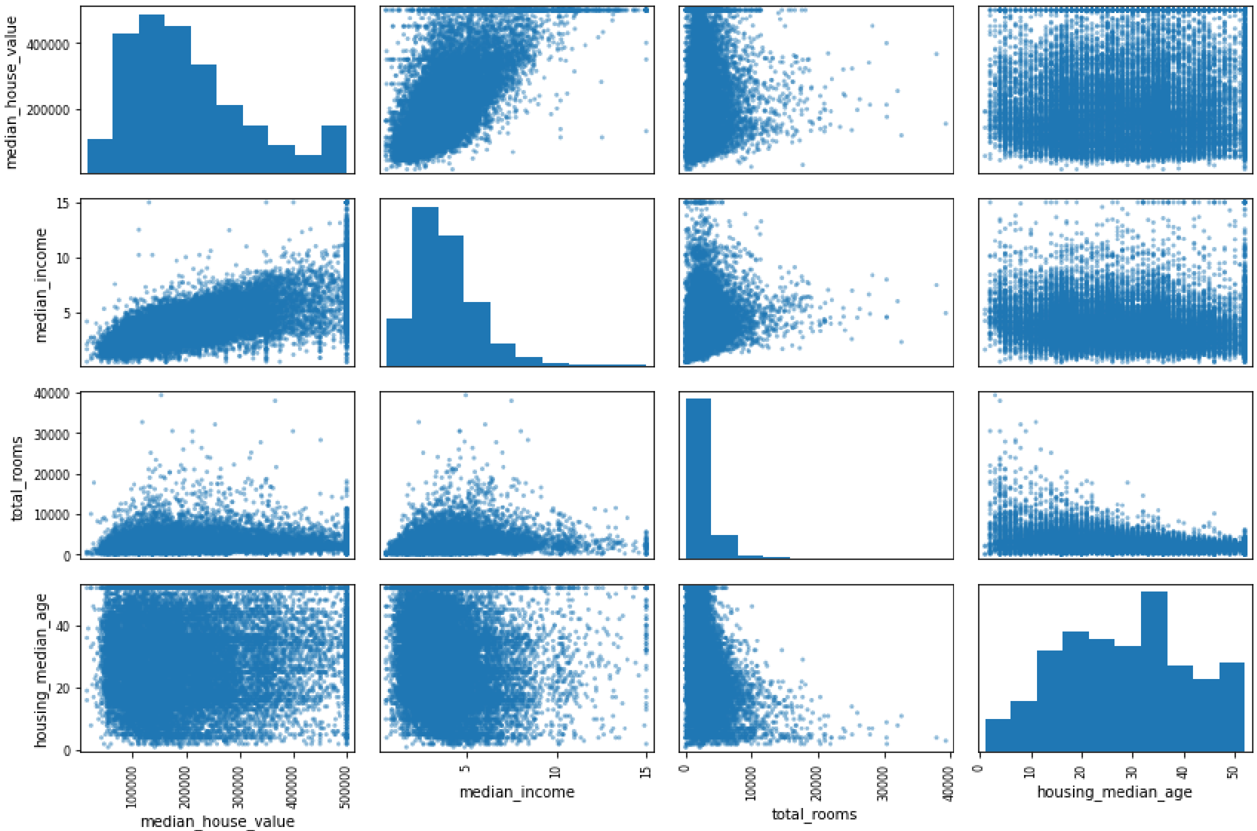
```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
```

```
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

In [27]:

```python
# the correlation matrix for different attributes/features can also be plotted
# some features may show a positive correlation/negative correlation or
# it may turn out to be completely random!
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot")
```

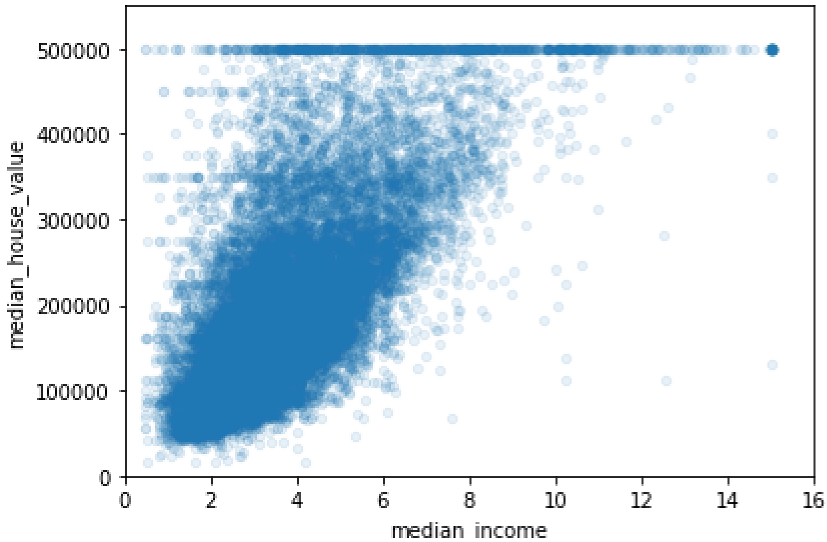
```
Saving figure scatter_matrix_plot
```



In [28]:

```python
# median income vs median house vlue plot plot 2 in the first row of top figure
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
save_fig("income_vs_house_value_scatterplot")
```

```
Saving figure income_vs_house_value_scatterplot
```



Augmenting Features

New features can be created by combining different columns from our data set.

- rooms_per_household = total_rooms / households
- bedrooms_per_room = total_bedrooms / total_rooms
- etc.

In [29]:

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

In [30]:

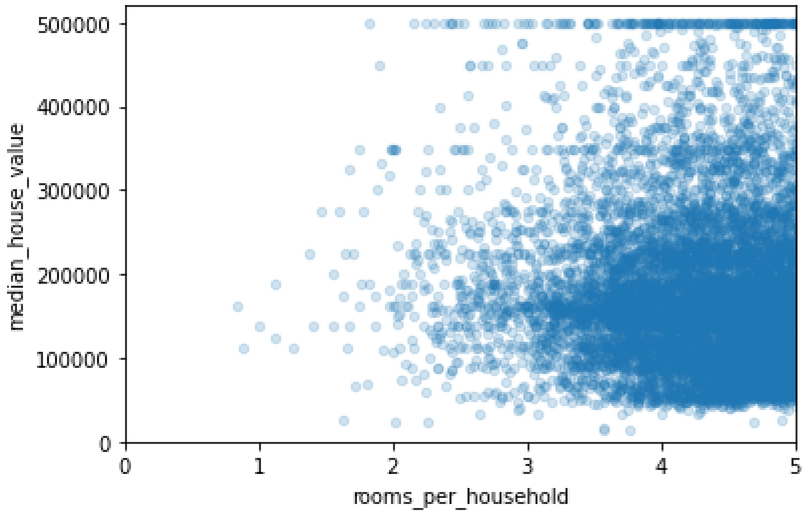
```
# obtain new correlations
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[30]:

```
median_house_value          1.000000
median_income               0.688075
rooms_per_household         0.151948
total_rooms                 0.134153
housing_median_age          0.105623
households                  0.065843
total_bedrooms              0.049686
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.255880
Name: median_house_value, dtype: float64
```

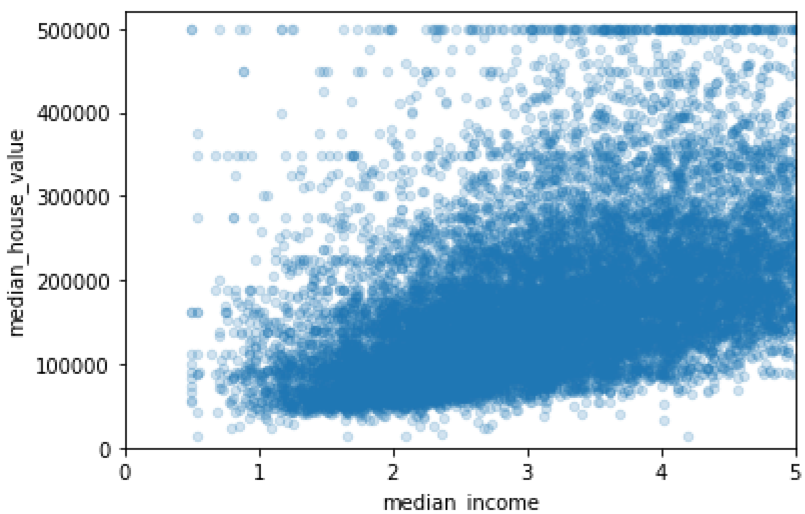
In [31]:

```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



In [32]:

```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



In [33]:

```
housing.describe()
```

Out[33]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000

Step 3. Preprocess the data for your machine learning algorithm

Once we've visualized the data, and have a certain understanding of how the data looks like. It's time to clean!

Most of your time will be spent on this step, although the datasets used in this project are relatively nice and clean... in the real world it could get real dirty.

After having cleaned your dataset you're aiming for:

- train set
- test set

In some cases you might also have a validation set as well for tuning hyperparameters (don't worry if you're not familiar with this term yet..)

In supervised learning setting your train set and test set should contain (**feature**, **target**) tuples.

- **feature**: is the input to your model
- **target**: is the ground truth label
  - when target is categorical the task is a classification task
  - when target is floating point the task is a regression task

We will make use of **scikit-learn** python package for preprocessing.

Scikit learn is pretty well documented and if you get confused at any point simply look up the function/object!

Dealing With Incomplete Data

In [34]:

```
# have you noticed when looking at the dataframe summary certain rows
# contained null values? we can't just leave them as nulls and expect our
# model to handle them for us so we'll have to devise a method for dealing with them...
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

Out[34]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	m
290	-122.16	37.77	47.0	1256.0	NaN	570.0	218.0	
341	-122.17	37.75	38.0	992.0	NaN	732.0	259.0	
538	-122.28	37.78	29.0	5154.0	NaN	3741.0	1273.0	
563	-122.24	37.75	45.0	891.0	NaN	384.0	146.0	
696	-122.10	37.69	41.0	746.0	NaN	387.0	161.0	

In [35]:

```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1: simply drop row
```

Out[35]:

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	medi

In [36]:

```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2: drop the comple
```

Out[36]:

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	m
290	-122.16	37.77	47.0	1256.0	570.0	218.0	4.3750	
341	-122.17	37.75	38.0	992.0	732.0	259.0	1.6196	
538	-122.28	37.78	29.0	5154.0	3741.0	1273.0	2.5762	
563	-122.24	37.75	45.0	891.0	384.0	146.0	4.9489	
696	-122.10	37.69	41.0	746.0	387.0	161.0	3.9063	

In [37]:

```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 3: repla
sample_incomplete_rows
```

Out[37]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	m
290	-122.16	37.77	47.0	1256.0	435.0	570.0	218.0	
341	-122.17	37.75	38.0	992.0	435.0	732.0	259.0	
538	-122.28	37.78	29.0	5154.0	435.0	3741.0	1273.0	
563	-122.24	37.75	45.0	891.0	435.0	384.0	146.0	
696	-122.10	37.69	41.0	746.0	435.0	387.0	161.0	

Could you think of another plausible imputation for this dataset? (Not graded)

Prepare Data

Recall we are trying to predict the median house value, our features will contain longitude, latitude, housing_median_age... and our target will be median_house_value

In [38]:

```
housing_features = housing.drop("median_house_value", axis=1) # drop labels for trainin
                                                        # the input to the model should
housing_labels = housing["median_house_value"].copy()
```

In [39]:

```
housing_features.head()
```

Out[39]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	mec
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

In [40]:

```
# This cell implements the complete pipeline for preparing the data
# using sklearns TransformerMixins
# Earlier we mentioned different types of features: categorical, and floats.
# In the case of floats we might want to convert them to categories.
# On the other hand categories in which are not already represented as integers must be
# feeding to the model.

# Additionally, categorical values could either be represented as one-hot vectors or si
# Here we encode them using one hot vectors.

# DO NOT WORRY IF YOU DO NOT UNDERSTAND ALL THE STEPS OF THIS PIPELINE. CONCEPTS LIKE N
# ONE-HOT ENCODING ETC. WILL ALL BE COVERED IN DISCUSSION

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin


imputer = SimpleImputer(strategy="median") # use median imputation for missing values
housing_num = housing_features.drop("ocean_proximity", axis=1) # remove the categorical
# column index
rooms_idx, bedrooms_idx, population_idx, households_idx = 3, 4, 5, 6

#
class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    implements the previous features we had defined
    housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
    housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
    housing["population_per_household"]=housing["population"]/housing["households"]
    '''
    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room

    def fit(self, X, y=None):
        return self  # nothing else to do

    def transform(self, X):
        rooms_per_household = X[:, rooms_idx] / X[:, households_idx]
        population_per_household = X[:, population_idx] / X[:, households_idx]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_idx] / X[:, rooms_idx]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
```

```
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = AugmentFeatures(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values) # generate new features

# this will be are numirical pipeline
# 1. impute, 2. augment the feature set 3. normalize using StandardScaler()
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeatures()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)

numerical_features = list(housing_num)
categorical_features = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

housing_prepared = full_pipeline.fit_transform(housing_features)
```

Splitting our dataset

First we need to carve out our dataset into a training and testing cohort. To do this we'll use train_test_split, a very elementary tool that arbitrarily splits the data into training and testing cohorts.

In [41]:

```
from sklearn.model_selection import train_test_split
data_target = housing['median_house_value']
train, test, target, target_test = train_test_split(housing_prepared, data_target, test
```

Select a model and train

Once we have prepared the dataset it's time to choose a model.

As our task is to predict the median_house_value (a floating value), regression is well suited for this.

In [42]:

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(train, target)

# let's try the full preprocessing pipeline on a few training instances
data = test
labels = target_test

print("Predictions:", lin_reg.predict(data)[:5])
print("Actual labels:", list(labels)[:5])
```

```
Predictions: [207828.06448011 281099.80175494 176021.36890539  93643.46744928
 304674.47047758]
```

```
Actual labels: [136900.0, 241300.0, 200700.0, 72500.0, 460000.0]
```

In [43]:

```
from sklearn.metrics import mean_squared_error

preds = lin_reg.predict(test)
mse = mean_squared_error(target_test, preds)
rmse = np.sqrt(mse)
rmse
```

Out[43]:

```
67879.86844243007
```

TODO: Applying the end-end ML steps to a different dataset.

We will apply what we've learnt to another dataset (airbnb dataset). We will predict airbnb price based on other features.

[35 pts] Visualizing Data

[5 pts] Load the data + statistics

- load the dataset
- display the first few rows of the data

In [78]:

```
def load_airbnb_data():
    csv_path = "./datasets/airbnb/AB_NYC_2019.csv"
    return pd.read_csv(csv_path)
airbnb = load_airbnb_data()
airbnb.head()
```

Out[78]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longituc
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.972
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.983
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.941
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.959
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.943

- pull up info on the data type for each of the data fields. Will any of these be problemmatic feeding into your model (you may need to do a little research on this)? Discuss:

In [79]:
```
airbnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              48895 non-null  int64  
 1   name                            48879 non-null  object 
 2   host_id                         48895 non-null  int64  
 3   host_name                       48874 non-null  object 
 4   neighbourhood_group             48895 non-null  object 
 5   neighbourhood                   48895 non-null  object 
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  object 
 9   price                           48895 non-null  int64  
 10  minimum_nights                  48895 non-null  int64  
 11  number_of_reviews               48895 non-null  int64  
 12  last_review                     38843 non-null  object 
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64  
 15  availability_365                48895 non-null  int64  
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

Yes. I can see some of these object types to be potentially problematic. neighborhood and heighbourhood_group are problematic and need to be converted to integers. Also room_type and last_review as well. last_review should be a datetime type but it is an object type so that needs to be converted. Also, things like id and host_id are not really relevent for our model and should be omitted.

- drop the following columns: name, host_id, host_name, and last_review
- display a summary of the statistics of the loaded data

In [80]:
```
airbnb_modified = airbnb.drop(['name', 'host_id', 'host_name', 'last_review'], axis=1)
```

In [81]:
```
airbnb_modified.describe()
```

Out[81]:

	id	latitude	longitude	price	minimum_nights	number_of_reviews	re
count	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	
mean	1.901714e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	
std	1.098311e+07	0.054530	0.046157	240.154170	20.510550	44.550582	
min	2.539000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	
25%	9.471945e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	
50%	1.967728e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	

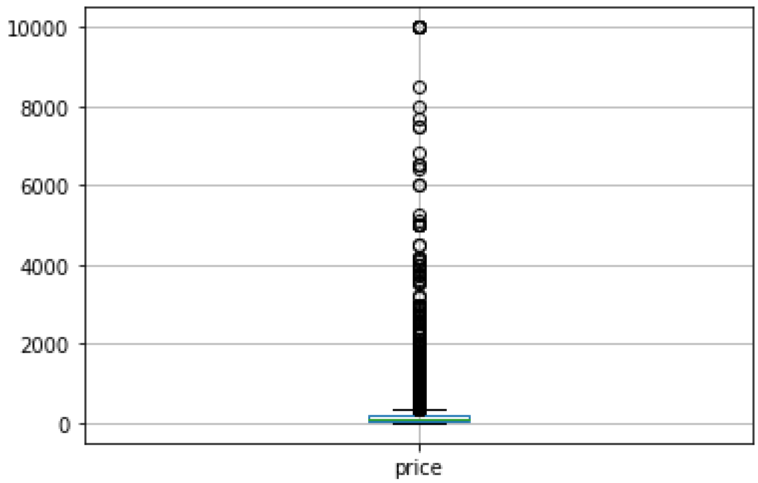
	id	latitude	longitude	price	minimum_nights	number_of_reviews	re
75%	2.915218e+07	40.763115	-73.936275	175.000000	5.000000	24.000000	
max	3.648724e+07	40.913060	-73.712990	10000.000000	1250.000000	629.000000	

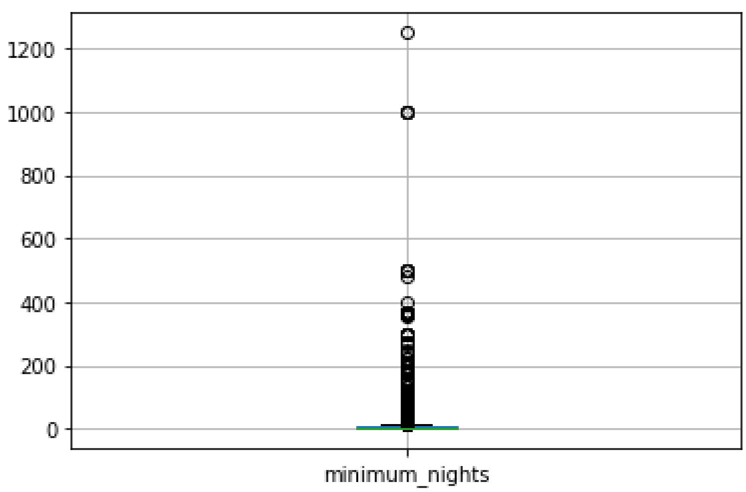
[5 pts] Boxplot 3 features of your choice

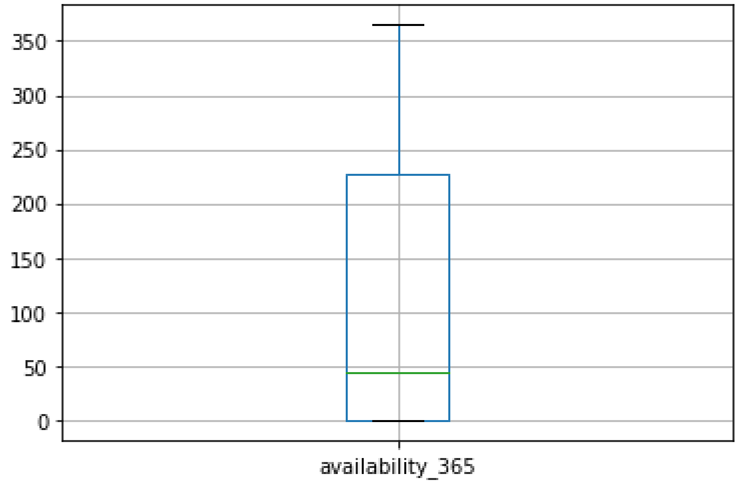
- plot boxplots for 3 features of your choice

In [82]:

```
boxplot_price = airbnb_modified.boxplot(column=['price'])
plt.show()
boxplot_nights = airbnb_modified.boxplot(column=['minimum_nights'])
plt.show()
boxplot_avail = airbnb_modified.boxplot(column=['availability_365'])
plt.show()
```







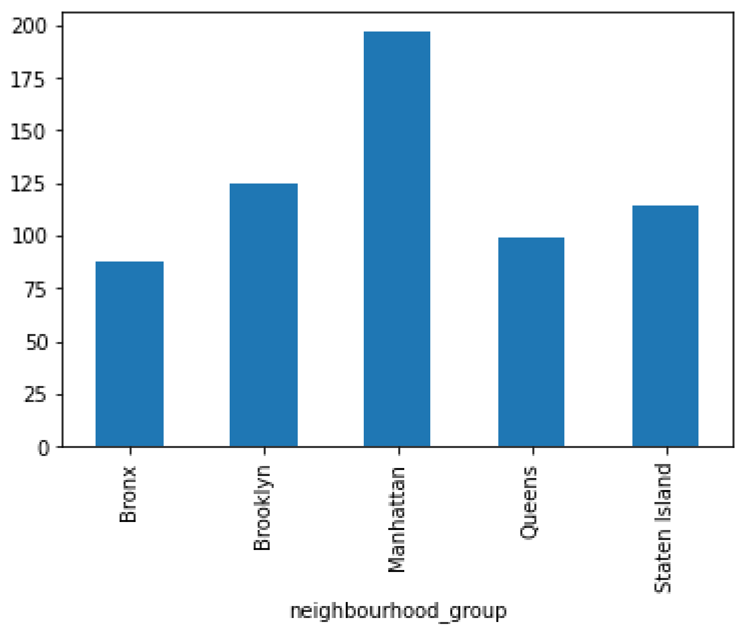
- describe what you expected to see with these features and what you actually observed

The price feature is highly variable and has a large skew. The same is true with minimum nights, I expected both of these to be less variable and skewed than they actually are. Availability is much more of a normal distribution, although it still has some skew, it is closer to what I would have expected.

High variability in price with long tail values, review numbers much more compact, however availability has a wider variance.

[10 pts] Plot average price of a listing per neighbourhood_group

In [83]:
```
airbnb.groupby("neighbourhood_group")['price'].mean().plot(kind="bar")
plt.show()
```



- describe what you expected to see with these features and what you actually observed

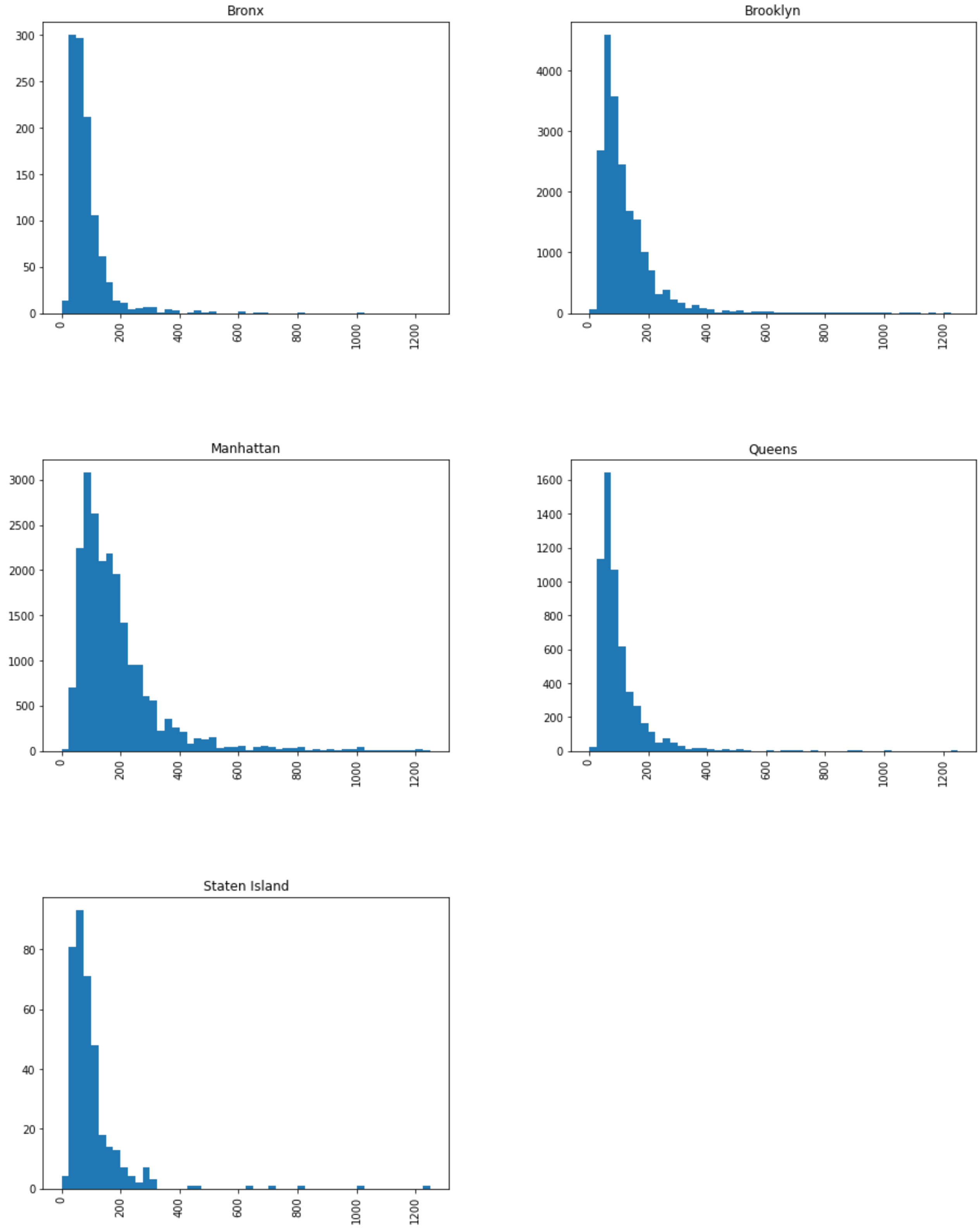
This is sort of what I expected to see. I am a little surprised at how varied the mean price per neighborhood is. I am not super familiar with NY however so I didn't expect much.

- So we can see different neighborhoods have dramatically different pricepoints, but how does the price breakdown by range. To see let's do a histogram of price by neighborhood to get a better sense of the distribution.

In [84]:

```
airbnb.hist(column='price', by="neighbourhood_group", bins=50, figsize=(15,20), range=(
plt.show()
```

```
/mnt/c/Users/ryanr/OneDrive/Desktop/UCLA/CSM148/lib/python3.7/site-packages/pandas/plott
ing/_matplotlib/tools.py:400: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed two minor
releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```

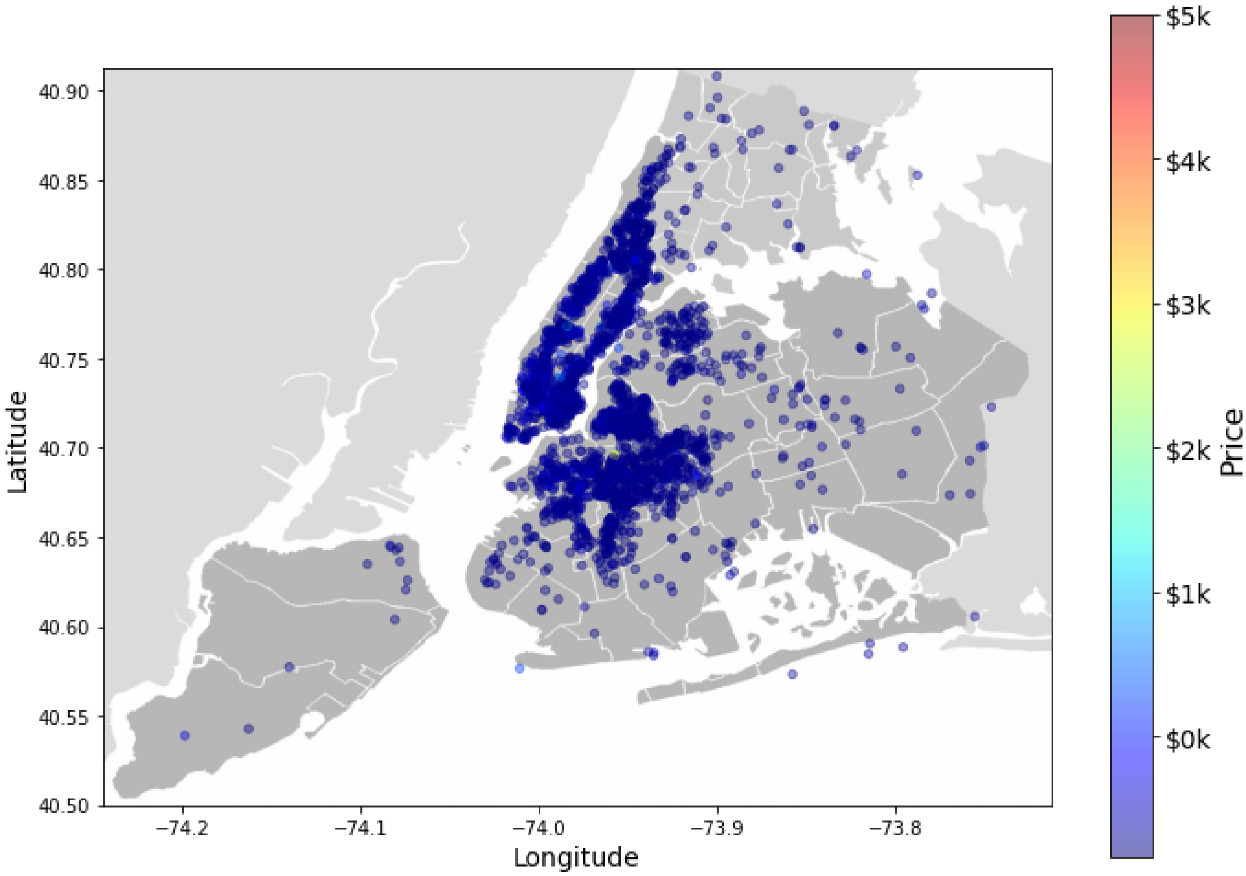


[5 pts] Plot map of airbnbs throughout New York (if it gets too crowded take a subset of the data, and try to make it look nice if you can :)).

In [85]:
```
import matplotlib.image as mpimg
newyork_img=mpimg.imread('./images/newyork.png')
ax = airbnb.iloc[3000:6000].plot(kind="scatter", x="longitude", y="latitude", figsize=(
                       #s=airbnb['population']/100, label="Population",
```

```
                        c="price", cmap=plt.get_cmap("jet"),
                        colorbar=False, alpha=0.4,
                       )
# overlay the new york map on the plotted scatter plot
# note: plt.imshow still refers to the most recent figure
# that hasn't been plotted yet.
plt.imshow(newyork_img, extent=[-74.24442, -73.71299, 40.49979, 40.91306], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

# setting up heatmap colors based on median_house_value feature
prices = airbnb["price"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cb = plt.colorbar()
cb.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14)
cb.set_label('Price', fontsize=16)

save_fig("new_york_airbnb_prices")
plt.show()
```

```
/mnt/c/Users/ryanr/OneDrive/Desktop/UCLA/CSM148/lib/python3.7/site-packages/ipykernel_la
uncher.py:20: UserWarning: FixedFormatter should only be used together with FixedLocator
Saving figure new_york_airbnb_prices
```

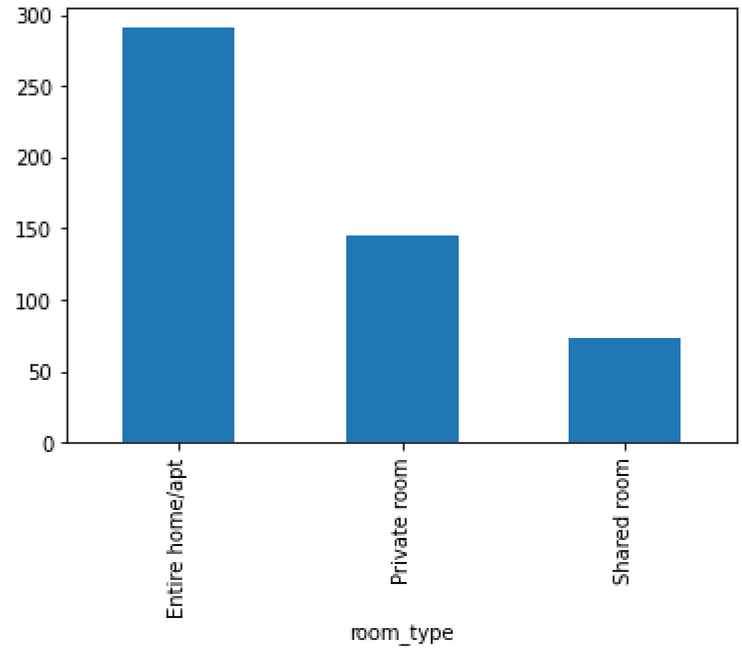


[5 pts] Plot average price of room types who have availability greater than 180 days and neighbourhood_group is Manhattan

In [86]:

```
airbnb.loc[(airbnb['availability_365'] > 180) & (airbnb['neighbourhood_group'] == 'Manh
```

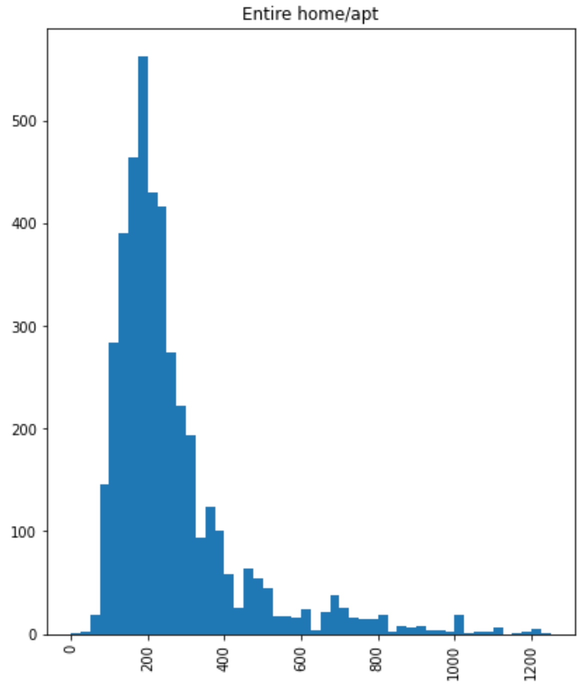
Out[86]: <AxesSubplot:xlabel='room_type'>

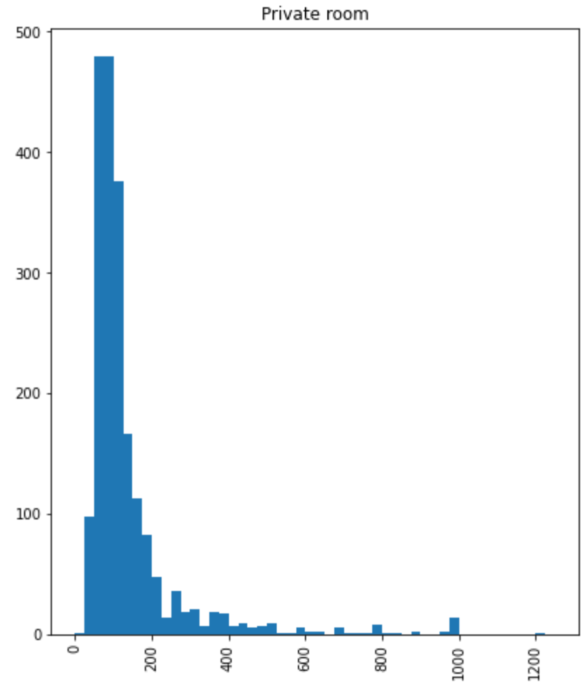


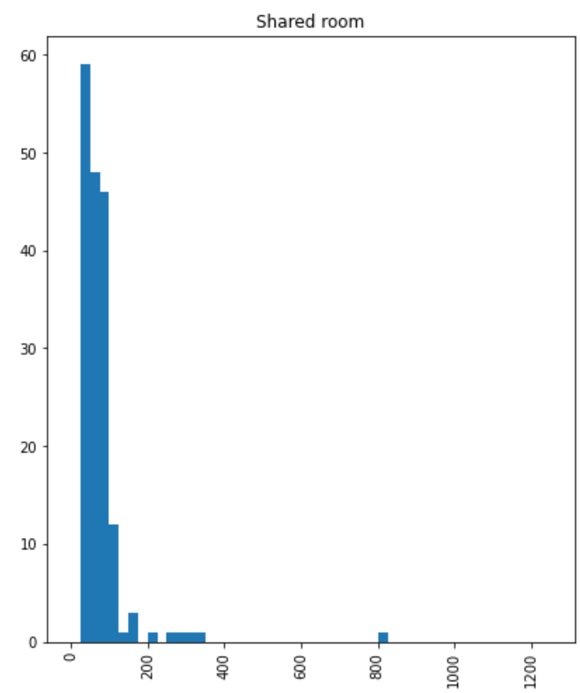
In [59]:

```
airbnb.loc[(airbnb['availability_365'] > 180) & (airbnb['neighbourhood_group'] == 'Manh
plt.show()
```

```
/mnt/c/Users/ryanr/OneDrive/Desktop/UCLA/CSM148/lib/python3.7/site-packages/pandas/plott
ing/_matplotlib/tools.py:400: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed two minor
releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```







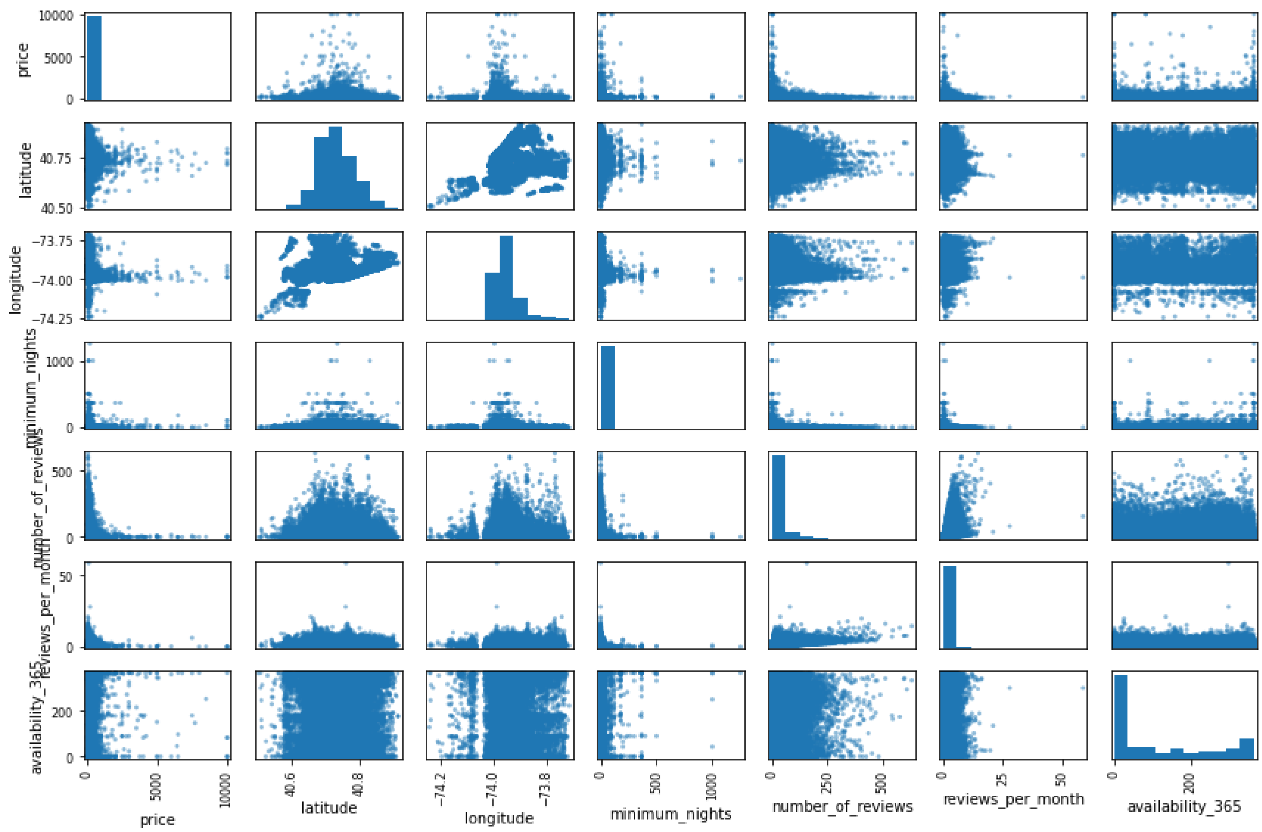
[5 pts] Plot correlation matrix

- which features have positive correlation?
- which features have negative correlation?

In [87]:
```
from pandas.plotting import scatter_matrix
attributes = ["price", "latitude", "longitude", "minimum_nights",
              "number_of_reviews", "reviews_per_month", "availability_365"]
```

```
scatter_matrix(airbnb_modified[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot_airbnb")
```

```
/mnt/c/Users/ryanr/OneDrive/Desktop/UCLA/CSM148/lib/python3.7/site-packages/pandas/plott
ing/_matplotlib/tools.py:400: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed two minor
releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
Saving figure scatter_matrix_plot_airbnb
```



Price appears to be negatively correlated with reviews_per_month, number_of_reviews and minimum_nights. minimum_nights is also negatively correlated with reviews_per_month.

reviews_per_month appears to be positively correlated with number_of_reviews.

Nothing seems very positivelty correlated, except latitude and longitude which makes sense given the shape of NY.

[30 pts] Prepare the Data

[5 pts] Augment the dataframe with two other features which you think would be useful

In [109…

```
airbnb['max_residents'] = airbnb['availability_365'] / airbnb['minimum_nights']
airbnb['reviews_per_availability'] = airbnb['number_of_reviews'] / airbnb['availability
```

[5 pts] Impute any missing feature with a method of your choice, and briefly discuss why you chose this imputation method

In [112…
```
#sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True)

airbnb_modified["reviews_per_month"].fillna(0, inplace=True)
'''
I chose to simply fill all the nan values for reviews_per_month with zeros
because I noticed that most of these were nan because the number_of_reviews field
happened to be zero, so to me it made sense to swap the nans with zeros.
'''
```

Out[112…
```
'\nI chose to simply fill all the nan values for reviews_per_month with zeros\nbecause I noticed that most of these were nan because the number_of_reviews field \nhappened to be zero, so to me it made sense to swap the nans with zeros.\n'
```

[15 pts] Code complete data pipeline using sklearn mixins

In [114…
```
airbnb_features = airbnb_modified.drop("price", axis=1) # drop labels for training set
                                                       # the input to the model should
airbnb_labels = airbnb_modified["price"].copy()

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin


imputer = SimpleImputer(strategy="median") # use median imputation for missing values
airbnb_num = airbnb_features.drop(["id", "neighbourhood_group", "neighbourhood", "room_
# column index
avail_idx, nights_idx, numreviews_idx = 7, 3, 4

#
class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    implements:
    airbnb['max_residents'] = airbnb['availability_365'] / airbnb['minimum_nights']
    airbnb['reviews_per_availability'] = airbnb['number_of_reviews'] / airbnb['availabi
    '''
    def __init__(self, add_max_res = True):
        self.add_max_res = add_max_res

    def fit(self, X, y=None):
        return self  # nothing else to do

    def transform(self, X):
        if (self.add_max_res):
            max_residents = X[:, avail_idx] / (X[:, nights_idx] + 1)
            reviews_per_availability = X[:, numreviews_idx] / (X[:, avail_idx] + 1)
            return np.c_[X, max_residents, reviews_per_availability]
        else:
            #reviews_per_availability = X[:, numreviews_idx] / X[:, avail_idx]
            return np.c_[X]

attr_adder = AugmentFeatures()
airbnb_extra_attribs = attr_adder.transform(airbnb_modified.values) # generate new feat
```

```
# this will be our numirical pipeline
# 1. impute, 2. augment the feature set 3. normalize using StandardScaler()
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeatures()),
        ('std_scaler', StandardScaler()),
    ])

airbnb_num_tr = num_pipeline.fit_transform(airbnb_num)

numerical_features = list(airbnb_num)
categorical_features = ["neighbourhood_group", "neighbourhood", "room_type"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

airbnb_prepared = full_pipeline.fit_transform(airbnb_features)
```

[5 pts] Set aside 20% of the data as test test (80% train, 20% test).

In [115…

```
from sklearn.model_selection import train_test_split
data_target = airbnb_modified['price']
train, test, target, target_test = train_test_split(airbnb_prepared, data_target, test_
```

[15 pts] Fit a model of your choice

The task is to predict the price, you could refer to the housing example on how to train and evaluate your model using MSE. Provide both test and train set MSE values.

In [120…

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lin_reg = LinearRegression()
lin_reg.fit(train, target)

# let's try the full preprocessing pipeline on a few training instances
data = test
labels = target_test

print("Predictions:", lin_reg.predict(data)[:5])
print("Actual labels:", list(labels)[:5])

preds_train = lin_reg.predict(train)
mse_train = mean_squared_error(target, preds_train)


preds_test = lin_reg.predict(test)
mse_test = mean_squared_error(target_test, preds_test)

print(f"MSE train: {mse_train}")
print(f"MSE test: {mse_test}")
```

```
Predictions: [544.76461775 317.14758756 191.77740824  67.81713865 148.83430067]
```

```
Actual labels: [225, 649, 300, 26, 125]
MSE train: 52591.269665996035
MSE test: 45980.18186988377
```